Deep Stack Fantasy

A Scheme for Removing EVM Stack Access Restrictions

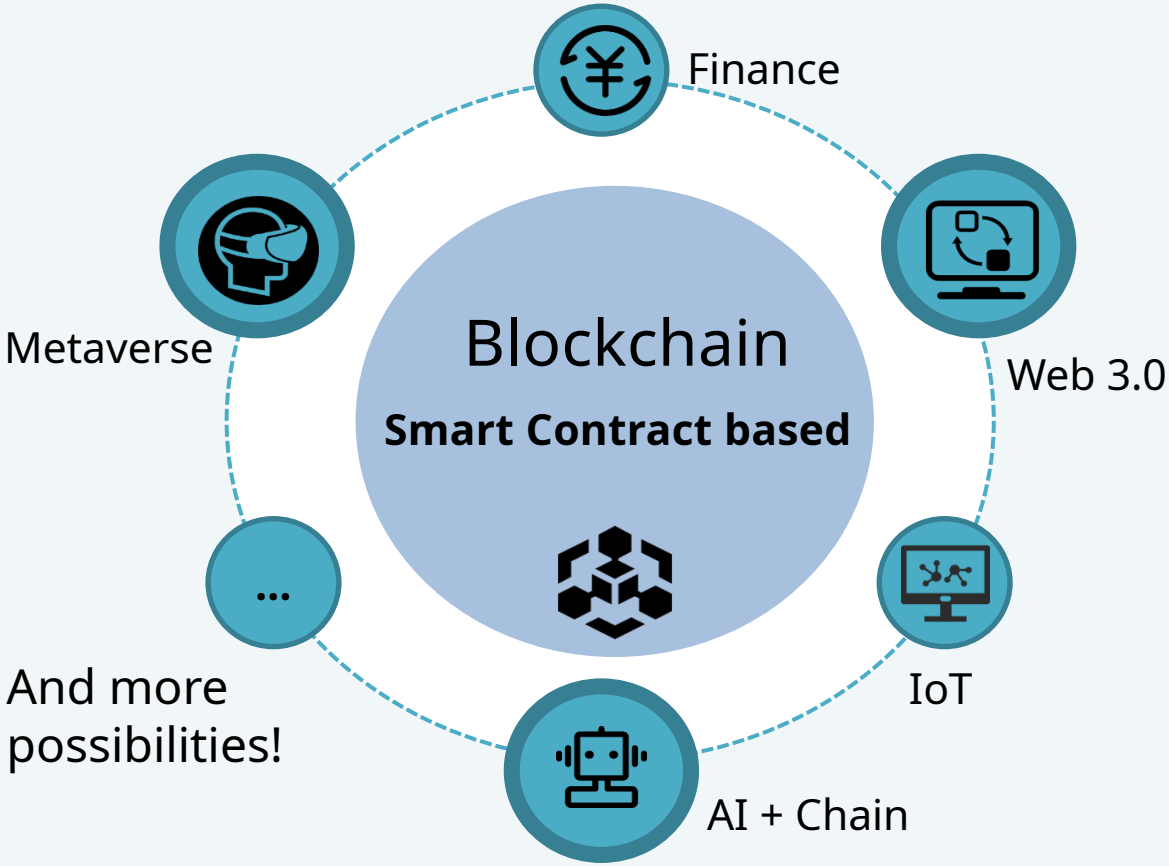
九转以太坊

GitHub Link: https://github.com/deep-stack-fantasy

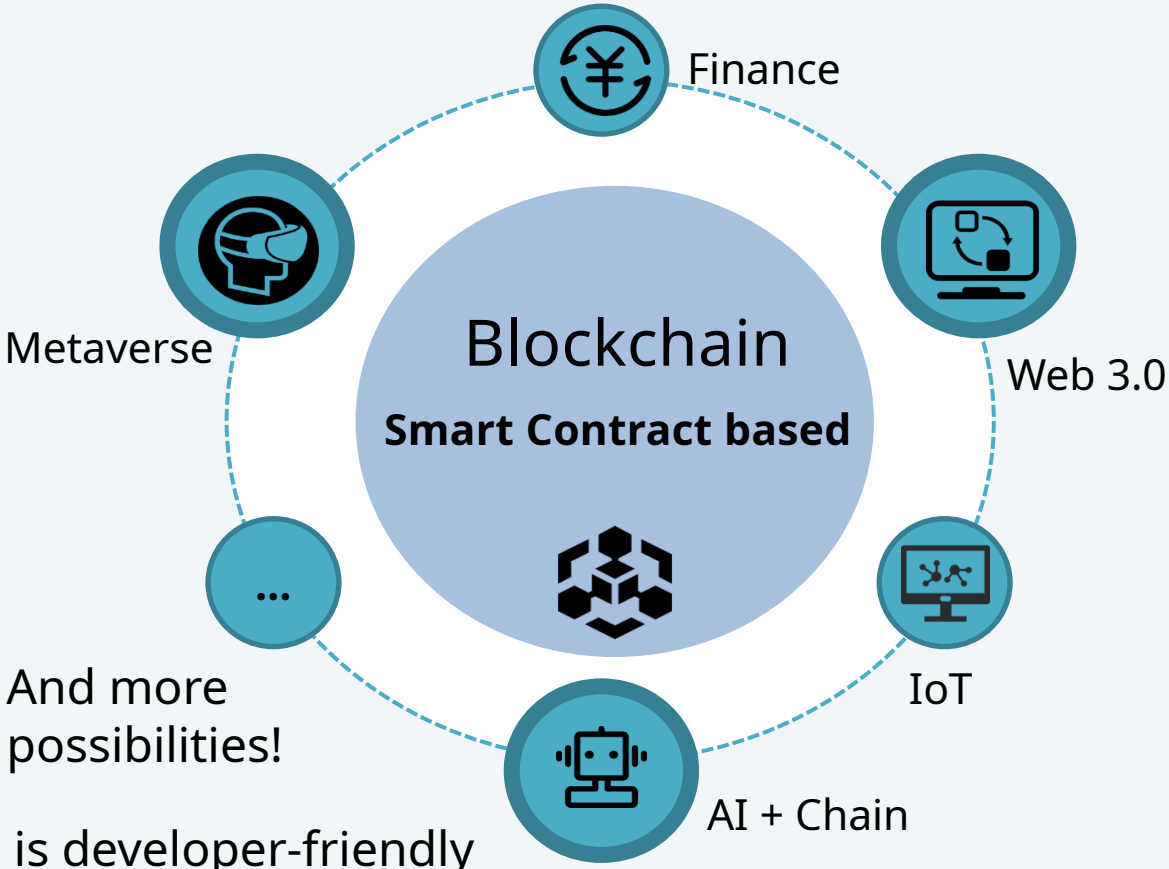
Deep Stack Fantasy

Note: pic from Internet.

Motivation: EVM applications keep growing



Motivation: EVM applications keep growing



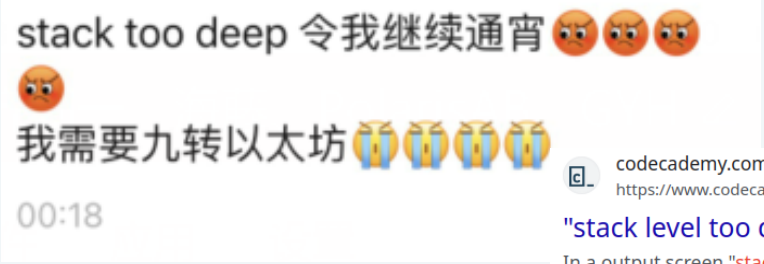
... if EVM is developer-friendly

Motivation

Which aspect of Solidity do you hate the most?

1. Debugging / Error handling
2. String support
3. Inheritance
4. JS-like syntax
5. “Stack too deep”
6. Arrays
7. Yul / Inline assembly
8. Mappings
9. Dealing with security
10. Tooling
11. Math in Solidity
12. Memory management
13. Modifiers
14. ABI encoder V2 too experimental

Data derived from 232 replies to the Solidity Summit registration form out of which 153 contained usable data for this question. 69% of the respondents answered with one of the Top 10 replies.



Note: 1st pic from https://soliditydeveloper.com/stacktoodeep.

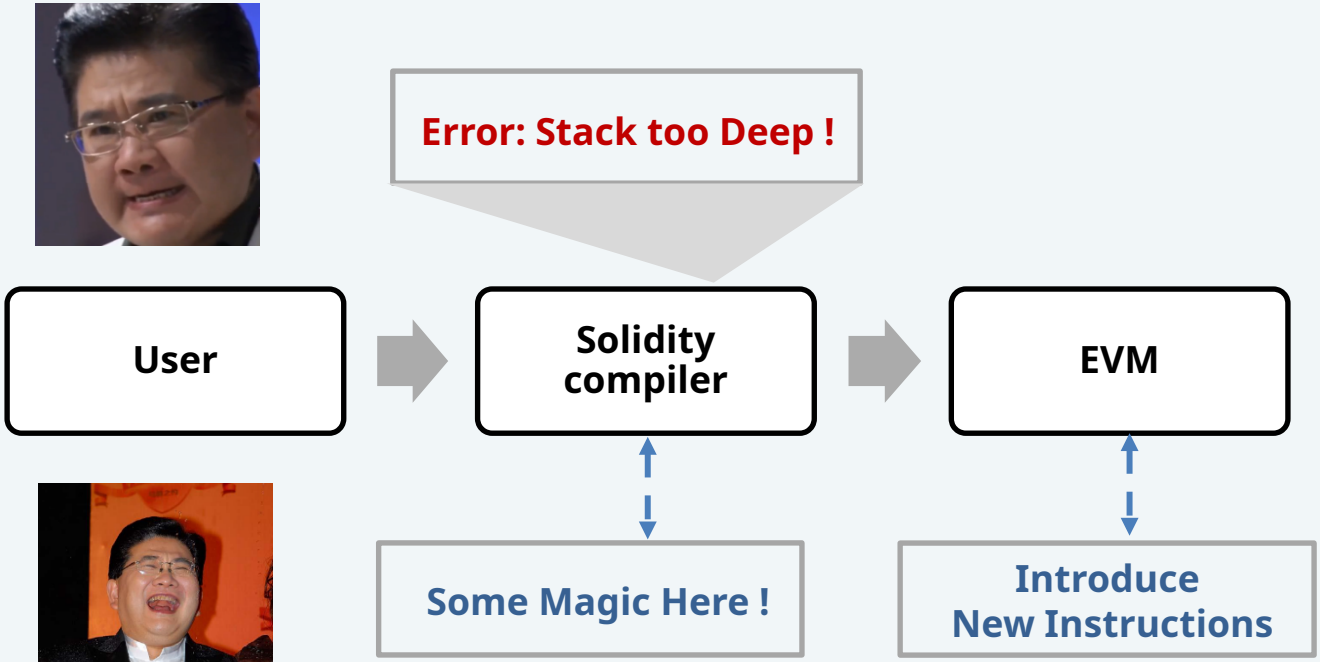
Problem

```
contract Test8 {

    function add_8(uint256 num1, uint256 num2,uint256 num3,
                   uint256 num4,uint256 num5,uint256 num6,
                       uint256 num7,uint256 num8)
    public view returns(uint256) {
        uint256 number = num1+num2+num3+num4+num5+num6+num7+num8;
        return number;
    }

}
```

```
CompilerError: Stack too deep, try
removing local variables. -->
```

Note: pics from Internet.

Our Work -- Overview



Note: pics from Internet.

Challenge

- Modifying the EVM instruction set -- a Harvard architecture.
  [Difficulty Level: Low]

- Instructions are variable-length.
  [Difficulty Level: High]

- Modifications to code generation. (A mess!)
  [Difficulty Level: High]

- Modifications to bytecode generation.
  [Difficulty Level: Medium]

- Debugging works, mess magic numbers, ... and time is running out!

Our Work -- Demo

```
contract Test8 {

    function add_8(uint256 num1, uint256 num2,uint256 num3,
                   uint256 num4,uint256 num5,uint256 num6,
                       uint256 num7,uint256 num8)
    public view returns(uint256) {
        uint256 number = num1+num2+num3+num4+num5+num6+num7+num8;
        return number;
    }

}
```

```
CompilerError: Stack too deep, try
removing local variables. -->
```

Original

BYTECODE


```
{
    "functionDebugData": {},
    "generatedSources": [],
    "linkReferences": {},
    "object": "608060405234a50115
    "opcodes": "PUSH1 0x80 PUSH1
    "sourceMap": "0:322:0:-:0;;;;
}
```


Our work

Our Work -- Demo

```
contract Test16 {

    function add_16(uint256 num1, uint256 num2,uint256 num3,
                    uint256 num4,uint256 num5,uint256 num6,
                    uint256 num7,uint256 num8,uint256 num9,
                    uint256 num10,uint256 num11, uint256 num12,
                    uint256 num13,uint256 num14,uint256 num15,uint256 num16)
    public view returns(uint256) {
        uint256 number = num1+num2+num3+num4+num5+num6+num7+num8+num9+
        num10+num11+num12+num13+num14+num15+num16;
        return number;
    }

}
```

BYTECODE


```
{
    "functionDebugData": {},
    "generatedSources": [],
    "linkReferences": {},
    "object": "608060405234a50115610011576000080fd5b506104498061002160003
    "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUPE ADD ISZERO P
    "sourceMap": "0:535:0:-:0;;;;;;;;;;;;;;;;;;;;"
}
```

Our Work -- Demo

```
contract Testa2z {
    function add_a2z(uint256 a, uint256 b, uint256 c, uint256 d, uint256 e, uint256 f, uint256 g,
        uint256 h, uint256 i, uint256 j, uint256 k, uint256 l, uint256 m, uint256 n,
        uint256 o, uint256 p, uint256 q, uint256 r, uint256 s, uint256 t,
        uint256 u, uint256 v, uint256 w, uint256 x, uint256 y, uint256 z)
        public pure returns (uint256) {
        return a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z;
    }
}
```

BYTECODE


```
{
    "functionDebugData": {},
    "generatedSources": [],
    "linkReferences": {},
    "object": "608060405234a501156100115760008 0fd5b506105d7806100216000:
    "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUPE ADD ISZERO P
    "sourceMap": "0:464:0:-:0;;;;;;;;;;;;;;;;;;;;"
}
```

Our Work -- Demo

```
contract Test100 {

    function add_100(uint256 num1, uint256 num2, uint256 num3,uint256 num4,uint25
    uint256 num11, uint256 num12, uint256 num13, uint256 num14, uint256 num15,uin
    uint256 num21, uint256 num22, uint256 num23, uint256 num24, uint256 num25, ui
    uint256 num31, uint256 num32, uint256 num33, uint256 num34, uint256 num35, ui
    uint256 num41, uint256 num42, uint256 num43, uint256 num44, uint256 num45, ui
    uint256 num51, uint256 num52, uint256 num53, uint256 num54, uint256 num55, ui
    uint256 num61, uint256 num62, uint256 num63, uint256 num64, uint256 num65, ui
    uint256 num71, uint256 num72, uint256 num73, uint256 num74, uint256 num75, ui
    uint256 num81, uint256 num82, uint256 num83, uint256 num84, uint256 num85, ui
    uint256 num91, uint256 num92, uint256 num93, uint256 num94, uint256 num95, ui
    public view returns(uint256){
        uint256 numbers = num1 + num2 + num3 + num4+num5+num6+num7+num8+num9+num1
        num23+num24+num25+num26+num27+num28+num29+num30+num31+num32+num33+num34+n
        num47+num48+num49+num50+num51+num52+num53+num54+num55+num56+num57+num58+n
        num71+num72+num73+num74+num75+num76+num77+num78+num79+num80+num81+num82+n
        num95+num96+num97+num98+num99+num100;
        return numbers;
    }

}
```

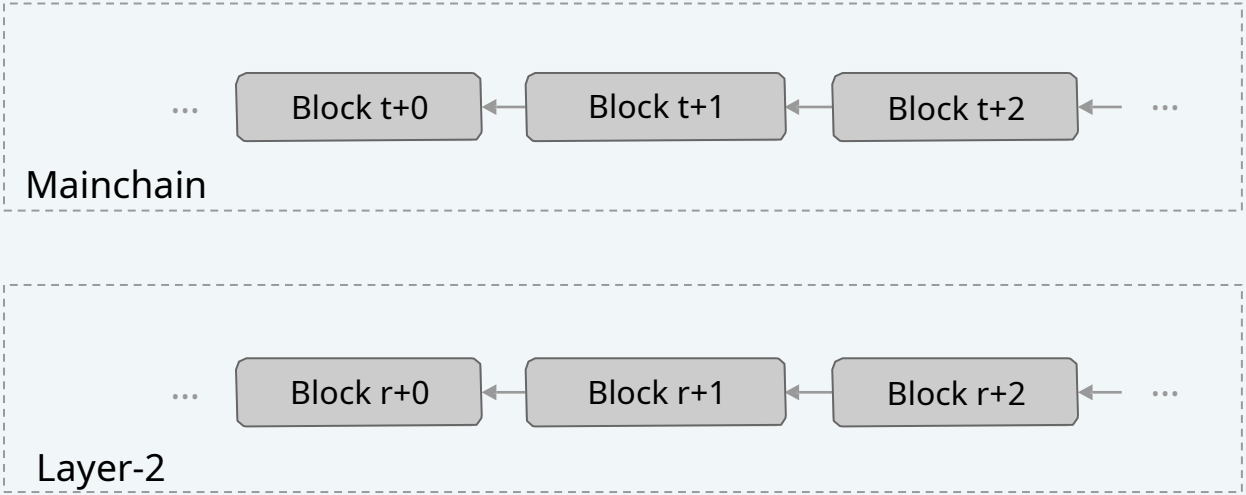
EVM incompatibility?

- Although appliciable in private chains, an extended EVM is incompatible with mainnet. :(

- Can we propose the extended EVM as a new EIP?
  - Requires performance evaluation as future works.

- Meanwhile,
  - our method can be applied to Layer-2 (zkEVM-based) schemes.

Apply to Layer-2?



zkEVM with
Deep Stack Fantasy

- Remove the access limitation on the EVM stack.
- Costs little to the mainchain.
- Also costs little to zk proof generation.

Future Work -- Apply to Scroll

dup.rs

zkevm-circuits > src > evm_circuit > execution > dup.rs

```rust
impl<F: Field> ExecutionGadget<F> for DupGadget<F> {
    const NAME: &'static str = "DUP";

    const EXECUTION_STATE: ExecutionState = ExecutionState::DUP;

    fn configure(cb: &mut ConstraintBuilder<F>) -> Self {
        let opcode = cb.query_cell();

        let value = cb.query_cell_phase2();

        // The stack index we have to peek, deduced from the 'x' value of 'dupx'
        // The offset starts at 0 for DUP1
        let dup_offset = opcode.expr() - OpcodeId::DUP1.expr();

        // Peek the value at `dup_offset` and push the value on the stack
        cb.stack_lookup(false.expr(), dup_offset, value.expr());
        cb.stack_push(value.expr());
```

Goal: Extend dup_offset from [1, 16] to [1, 255]

An Engineer from Scroll

The relationship between **stack pointer offset** and **the size of zero-knowledge proof circuit** is constant in zkEVM.

Future Work -- Apply to Scroll

constraint_builder.rs

zkevm-circuits > src > evm_circuit > util > constraint_builder.rs

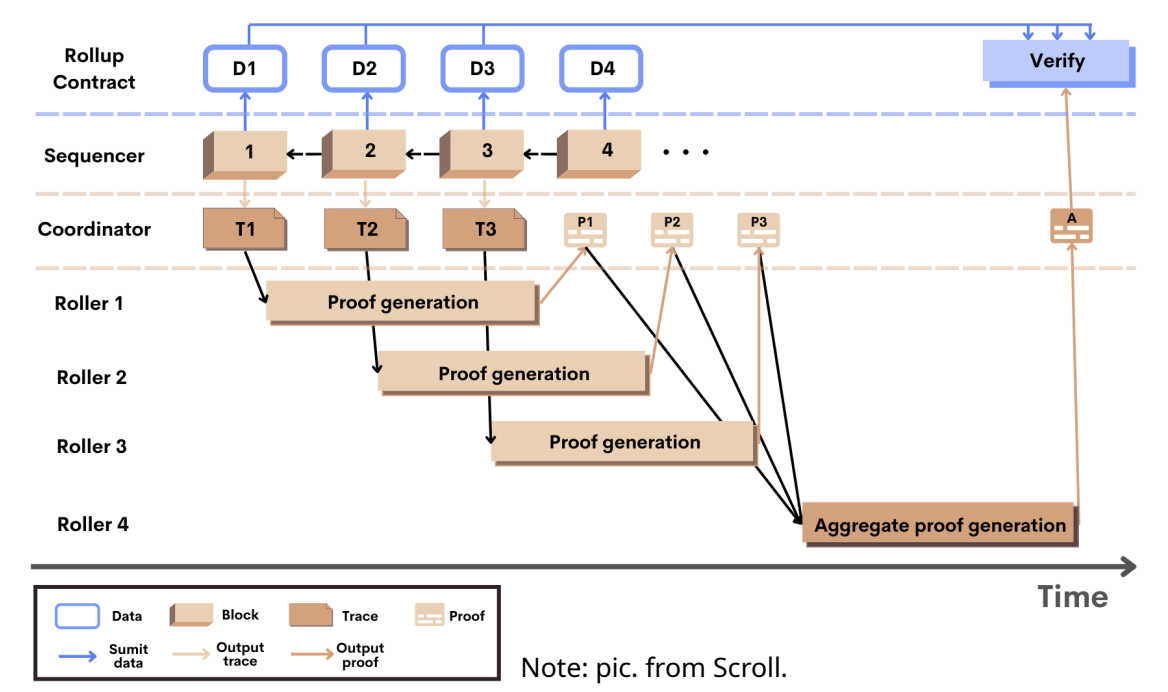
```rust
pub(crate) fn stack_lookup(
    &mut self,
    is_write: Expression<F>,
    stack_pointer_offset: Expression<F>,
    value: Expression<F>,
) {
    self.rw_lookup(
        "Stack lookup",
        is_write,
        RwTableTag::Stack,
        RwValues::new(
            self.curr.state.call_id.expr(),
            self.curr.state.stack_pointer.expr() + stack_pointer_offset
            0.expr(),
```

i.e., Extend stack_pointer_offset from [1, 16] to [1, 255]

An Engineer from Scroll

The relationship between **stack pointer offset** and **the size of zero-knowledge proof circuit** is constant in zkEVM.

Future Work -- Apply to Scroll



Note: pic. from Scroll.

An Engineer from Scroll

The running requirement for Scroll nodes (generating proofs) is at least 500 GB RAM → leave it as a future work

Deep Stack Fantasy

Growing ETH community by making devs' life easier